**SQL Aggregate Functions**

## Introduction to SQL aggregate functions

An aggregate function allows you to perform a calculation on a set of values to return a single scalar value. We often use aggregate functions with the [GROUP BY](http://www.zentut.com/sql-tutorial/sql-group-by/) and [HAVING](http://www.zentut.com/sql-tutorial/sql-having/) clauses of the [SELECT](http://www.zentut.com/sql-tutorial/sql-select/) statement.

The following are the most commonly used SQL aggregate functions:

* [**AVG**](http://www.sqltutorial.org/sql-avg.aspx) **– calculates the average of a set of values.**
* [**COUNT**](http://www.sqltutorial.org/sql-count.aspx) **– counts rows in a specified table or view.**
* [**MIN**](http://www.sqltutorial.org/sql-min-max.aspx) **– gets the minimum value in a set of values.**
* [**MAX**](http://www.zentut.com/sql-tutorial/sql-max/) **– gets the maximum value in a set of values.**
* [**SUM**](http://www.sqltutorial.org/sql-sum.aspx) **– calculates the sum of values.**

Notice that all aggregate functions above ignore NULL values except for the COUNT function.

## **SQL aggregate functions syntax**

To call an aggregate function, you use the following syntax:

|  |  |
| --- | --- |
| 1 | **aggregate\_function (DISTINCT | ALL expression)** |

Let’s examine the syntax above in greater detail:

* First, specify an aggregate function that you want to use e.g., MIN, MAX, AVG, SUM or COUNT.
* Second, put DISTINCT or ALL modifier followed by an expression inside parentheses. If you explicitly use DISTINCT modifier, the aggregate function ignores duplicate values and only consider the unique values. If you use the ALL modifier, the aggregate function uses all values for calculation or evaluation. The  ALL modifier is used by default if you do not specify any modifier explicitly.

## **SQL aggregate function examples**

Let’s take a look some examples of using SQL aggregate functions.

### **COUNT function example**

To get the number of the products in the products table, you use the COUNT function as follows:

|  |  |
| --- | --- |
| 1  2  3  4 | **SELECT**  **COUNT(\*)**  **FROM**  **products;** |

SQL COUNT function example

### **AVG function example**

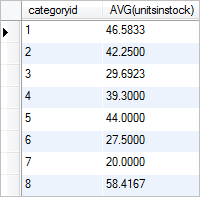
To calculate the average units in stock of the products, you use the AVG function as follows:

|  |  |
| --- | --- |
| 1  2  3  4 | **SELECT**  **AVG(unitsinstock)**  **FROM**  **products;** |

SQL avg units in stock

To calculate units in stock by product category, you use the AVG function with the  GROUP BY clause as follows:

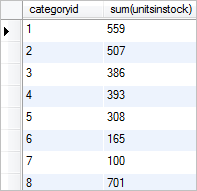
|  |  |
| --- | --- |
| 1  2  3  4  5 | **SELECT**  **categoryid, AVG(unitsinstock)**  **FROM**  **products**  **GROUP BY categoryid;** |



### **SUM function example**

To calculate the sum of units in stock by product category, you use the SUM function with the  GROUP BY clause as the following query:

|  |  |
| --- | --- |
| 1  2  3  4  5 | **SELECT**  **categoryid, SUM(unitsinstock)**  **FROM**  **products**  **GROUP BY categoryid;** |



### MIN function example

To get the minimum units in stock of products in the products table, you use the MIN function as follows:

|  |  |
| --- | --- |
| 1  2  3  4 | **SELECT**  **MIN(unitsinstock)**  **FROM**  **products;** |

SQL MIN example

**MAX function example**

To get the maximum units in stock of products in the products table, you use the MAX function as shown in the following query:

|  |  |
| --- | --- |
| 1  2  3  4 | **SELECT**  **MAX(unitsinstock)**  **FROM**  **products;** |

SQL MAX function example

# SQL AVG Function

## Introduction to SQL AVG function

The AVG function calculates the average of the values. To use the AVG function, you use the following syntax:

|  |  |
| --- | --- |
| 1 | **AVG (ALL | DISTINCT expression)** |

You can specify ALL or DISTINCT modifier before the expression.

* ALL modifier means that the AVG function is applied to all values including duplicates. The  AVG() function uses the ALL modifier by default if you do not specify any modifier explicitly.
* DISTINCT modifier means that the AVG function is applied to only distinct values in the set of values.

Notice that the AVG function ignores NULL values.

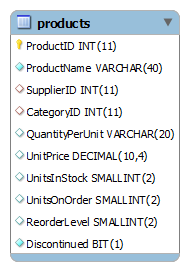
The AVG function returns a single value whose data type is determined by the type of the result of the expression. The returned data type could be any numeric type such as integer, float, etc.

Behind the scenes, the AVG function calculates the average of values by dividing the total of these values by the number of values except for the NULL values. Therefore, if the total of those values exceeds the maximum value of data type of the result, the database server will issue an error.

The AVG function is one of an ANSI [SQL aggregate functions](http://www.zentut.com/sql-tutorial/sql-aggregate-functions/), therefore, it is available in all relational database management systems e.g., Oracle, Microsoft SQL Server, MySQL, PostgreSQL, etc.

## **SQL AVG function examples**

We will use the products table in the [sample database](http://www.zentut.com/sql-tutorial/sql-sample-database/) for the demonstration of the  AVG() function in the following sections.

[](http://www.zentut.com/wp-content/uploads/2012/10/products.png)

### **Simple SQL AVG function example**

The following query calculates the average of unit prices of all products in the products table.

|  |  |
| --- | --- |
| 1  2  3  4 | SELECT      AVG(unitprice)  FROM      products; |

SQL AVG example

The query calculates the total unit prices and divides the total by the number of rows in the products table.

To calculate the average of distinct unit prices of products, you can use the DISTINCT modifier in the  AVG() function as the following query:

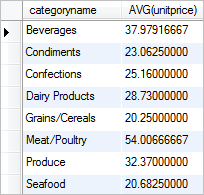
|  |  |
| --- | --- |
| 1  2  3  4 | **SELECT**  **AVG(DISTINCT unitprice)**  **FROM**  **products;** |

### SQL AVG DISTINCT example

### **SQL AVG function with GROUP BY clause**

To find the average of unit prices for each product’s category, you can use the AVG function with the [GROUP BY](http://www.zentut.com/sql-tutorial/sql-group-by/) clause as the following query:

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7** | **SELECT**  **categoryname, AVG(unitprice)**  **FROM**  **products**  **INNER JOIN**  **categories ON categories.categoryid = products.categoryid**  **GROUP BY categoryname;** |

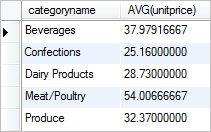


The [INNER JOIN](http://www.zentut.com/sql-tutorial/sql-inner-join/) clause is used to get the category name from the categories table.

### **SQL AVG function with HAVING clause**

To get the category that has an average unit price greater than $25, you use the AVG function with  GROUP BY and [HAVING](http://www.zentut.com/sql-tutorial/sql-having/) clauses as the following query:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | **SELECT**  **categoryname, AVG(unitprice)**  **FROM**  **products**  **INNER JOIN**  **categories ON categories.categoryid = products.categoryid**  **GROUP BY categoryname**  **HAVING AVG(unitprice) > 25;** |



# SQL COUNT Function

## **Introduction to SQL COUNT function**

The COUNT() function returns the number of rows in a group. The first form of the COUNT()function is as follows:

|  |  |
| --- | --- |
| 1 | **COUNT(\*)** |

The  COUNT(\*) function returns a number of rows in a specified table or view that includes the number of duplicates and NULL values.

To return the number of rows that excludes the number of duplicates and NULL values, you use the following form of the  COUNT() function:

|  |  |
| --- | --- |
| 1 | **COUNT(DISTINCT column)** |

To return the number of rows that includes the number of duplicates and excludes the number of the NULL values, you use the following form of the  COUNT() function:

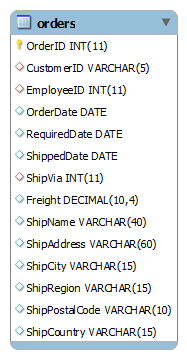
|  |  |
| --- | --- |
| 1 | **COUNT(ALL column)** |

The following table illustrates all forms of the  COUNT() function:

| **COUNT() Function** | **Count Duplicates** | **Count NULL values** |
| --- | --- | --- |
| COUNT(\*) | Yes | Yes |
| COUNT(DISTINCT column) | No | No |
| COUNT(ALL column) | Yes | No |

## **SQL COUNT(\*) function examples**

We will use the orders table in the [sample database](http://www.zentut.com/sql-tutorial/sql-sample-database/) in the following  COUNT(\*) function examples.



### **Simple SQL COUNT(\*) example**

To get the number of orders in the orders table, you use the  COUNT(\*) function as follows:

|  |  |
| --- | --- |
| 1  2  3  4 | **SELECT**  **COUNT(\*)**  **FROM**  **orders;** |

SQL COUNT example

The pending order is the order whose shipped date is NULL. To get the number of pending orders, you use the following query:

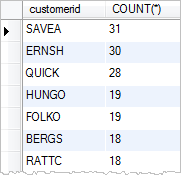
|  |  |
| --- | --- |
| 1  2  3  4  5  6 | **SELECT**  **COUNT(\*) 'Pending orders'**  **FROM**  **orders**  **WHERE**  **shippeddate IS NULL;** |

SQL COUNT with WHERE clause

### **SQL COUNT(\*) with GROUP BY clause example**

To get the number of orders by customers, you use the  COUNT(\*) function with the [GROUP BY](http://www.zentut.com/sql-tutorial/sql-group-by/) clause as the following query:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | **SELECT**  **customerid, COUNT(\*)**  **FROM**  **orders**  **GROUP BY customerid**  **ORDER BY COUNT(\*) DESC;** |

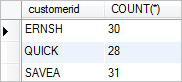


The  GROUP BY clause is used to group the orders by customers. For each group, the  COUNT(\*) function counts the orders by customer.

### **SQL COUNT(\*) with HAVING clause example**

To get customers who have more than 20 orders, you use the  COUNT(\*) function with  GROUP BY and [HAVING](http://www.zentut.com/sql-tutorial/sql-having/) clauses as the following query:

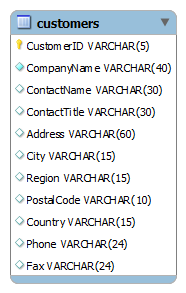
|  |  |
| --- | --- |
| 1  2  3  4  5  6 | **SELECT**  **customerid, COUNT(\*)**  **FROM**  **orders**  **GROUP BY customerid**  **HAVING COUNT(\*) > 20;** |



The  GROUP BY clause divides the orders into groups by customerid. The  COUNT(\*) function returns the number of orders for each customerid. The HAVING clause gets only groups that have more than 20 orders.

## **SQL COUNT ALL example**

Let’s take a look at the customers table.



To count all customers, you use the following query:

|  |  |
| --- | --- |
| 1  2  3  4 | **SELECT**  **COUNT(\*)**  **FROM**  **customers;** |

SQL COUNT customers

The following query returns the number of countries except for the NULL values:

|  |  |
| --- | --- |
| 1  2  3  4 | **SELECT**  **COUNT(ALL country)**  **FROM**  **customers;** |

SQL COUNT ALL example

## **SQL COUNT DISTINCT example**

To exclude both NULL values and duplicates, you use the  COUNT(DISTINCT column) as the following query:

|  |  |
| --- | --- |
| 1  2  3  4 | **SELECT**  **COUNT(DISTINCT country)**  **FROM**  **customers;** |

SQL COUNT DISTINCT example

# SQL MIN Function

## **Introduction to SQL MIN function**

The MIN function returns the minimum value in a set of values. The MIN function ignores the NULL values. The following is the syntax of the MIN function:

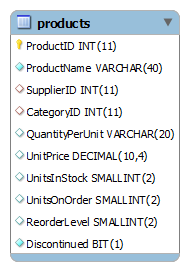
|  |  |
| --- | --- |
| 1 | **MIN(DISTINCT or ALL expression)** |

The ALL modifier instructs the MIN function to find the minimum value in all values including duplicates. The MIN() function uses the ALL modifier by default so you don’t have to specify it explicitly.

Unlike other [aggregate functions](http://www.zentut.com/sql-tutorial/sql-aggregate-functions/) e.g., [SUM](http://www.zentut.com/sql-tutorial/sql-sum/), [COUNT,](http://www.zentut.com/sql-tutorial/sql-count/) and [AVG](http://www.zentut.com/sql-tutorial/sql-avg/), the DISTINCT modifier is not applicable to the MIN() function. The DISTINCT modifier is only for ISO compatibility.

## **SQL MIN function examples**

Let’s take a look at the products table in the [sample database](http://www.zentut.com/sql-tutorial/sql-sample-database/):



### **Simple MIN function example**

To find the lowest unit price of products in the products table, you use the MIN() function as follows:

|  |  |
| --- | --- |
| 1  2  3  4 | **SELECT**  **MIN(unitprice)**  **FROM**  **products;** |

SQL MIN example

To get the cheapest products, you have to use a [subquery](http://www.zentut.com/sql-tutorial/sql-subquery/) that uses the MIN() function as the following query:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | **SELECT**  **productid, productname, unitprice**  **FROM**  **products**  **WHERE**  **unitprice = (**  **SELECT**  **MIN(unitprice)**  **FROM**  **products);** |

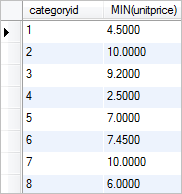
### SQL MIN subquery

The outer query gets the cheapest products whose unit prices match the lowest price returned by the subquery. If multiple products have the same unit price as the lowest price, the query will return more than one row.

### **SQL MIN function with GROUP BY clause example**

To find the lowest unit price of the product in each category, you use the MIN() function with a [GROUP BY](http://www.zentut.com/sql-tutorial/sql-group-by/) clause:

|  |  |
| --- | --- |
| 1  2  3  4  5 | **SELECT**  **categoryid, MIN(unitprice)**  **FROM**  **products**  **GROUP BY categoryid;** |

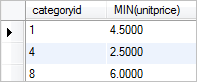


The GROUP BY clause divides the products by categoryid into groups. For each group, the MIN() function returns the lowest unit price.

### **SQL MIN function with HAVING clause example**

To get the category that has the lowest unit price less than $7, you use the MIN() function with the GROUP BY and [HAVING](http://www.zentut.com/sql-tutorial/sql-having/) clauses as follows:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | **SELECT**  **categoryid, MIN(unitprice)**  **FROM**  **products**  **GROUP BY categoryid**  **HAVING MIN(unitprice) < 7;** |



### **SQL MIN with correlated subquery example**

To get the cheapest product in each category, you use the  MIN() function in a [correlated subquery](http://www.zentut.com/sql-tutorial/understanding-correlated-subquery/) as follows:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | **SELECT categoryid,**  **productid,**  **productName,**  **unitprice**  **FROM products a**  **WHERE unitprice = (**  **SELECT MIN(unitprice)**  **FROM products b**  **WHERE b.categoryid = a.categoryid)** |

The outer query scans all rows in the products table and returns the products that have unit prices match the lowest price in each category returned by the correlated subquery.